MATHEMATICAL INSTITUTE

UNIVERSITY OF OXFORD

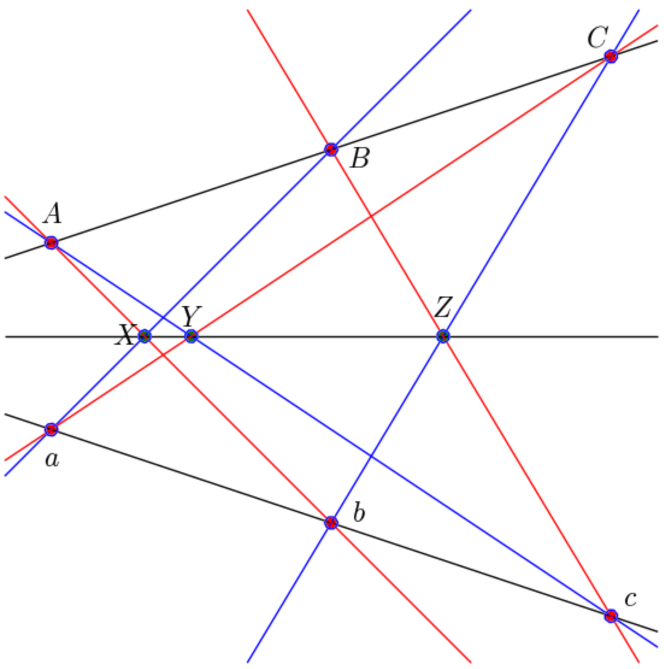
Computational Mathematics

Students' Guide

Hilary Term 2017

by

Dr Andrew Thompson



Acknowledgments: My thanks go to Dr Dmitrii Pasechnik who largely wrote the Sage project. The course guide is also based on previous versions by Dr Colin Macdonald and others before him. I am grateful to all previous authors.

Contents

Chapter 1

Introduction

The use of computers is widespread in all areas of life, and at universities they are used in both teaching and research. The influence and power of computing is fundamentally affecting many areas of both applied and pure mathematics. MATLAB is one of several systems used at Oxford for doing mathematics by computer; others include Mathematica, Maple, Sage and SciPy/NumPy. These tools are sufficiently versatile to support many different branches of mathematical activity, and they may be used to construct complicated programs.

1.1 Objectives

The objective of this practical course is to discover more about mathematics using MATLAB . Last term you were introduced to some basic techniques, by working through the Michaelmas Term Students' Guide which you are due to complete near the beginning of this term. After this, for the rest of Hilary Term, you will work alone on two projects. You will need to submit two projects from those in this manual.

While MATLAB complements the traditional part of the degree course, we hope the projects help you revise or understand topics which are related in some way to past or future lectures. It is hoped that at the end of this MATLAB course you will feel sufficiently confident to be able to use MATLAB (and/or other computer tools) throughout the rest of your undergraduate career.

1.2 Schedule

Deadlines

- 12 noon, Monday, week 6 (February 20) Submission of first project.
- 12 noon, Monday, week 9 (March 13) Submission of second project.

Computer and demonstrator access

This term, the practical sessions with demonstrators in Weeks 1 and 2 will be the same as those for weeks 7 and 8, respectively, of Michaelmas Term. From Week 3 onwards, there are no fixed hours for each college, and you may use the timetabled classrooms at the Mathematical Institute whenever suits you within the following times:

- Weeks 3–8: Mon 3pm–4pm; Thurs 3pm–4pm.

There will be additional sessions ahead of the project deadlines:

- Week 5: Weds 3pm–5pm; Fri 3pm–5pm.
- Week 8: Fri 3pm–5pm.

It's probably a good idea to check the course website (`https://courses.maths.ox.ac.uk/node/30`) for any possible updates to these times.

Note that you will need to bring your laptop to the drop-in sessions. If you need to borrow a laptop, you will need to inform Nia Roderick (`roderick@maths.ox.ac.uk`) of your chosen drop-in session times in advance.

A demonstrator will be present during the above times. Demonstrators will help resolve general problems that you encounter in trying to carry out the instructions of this booklet, but will not assist in the actual project exercises.

1.3 Completing the projects

To carry out a project successfully, you need to master two ingredients—the actual mathematics of the topic under investigation, and the construction of the MATLAB commands needed to solve the relevant problems. Picking up the mathematics is probably a familiar activity that you practice when you attend a lecture or read your notes. Building up a repertoire of MATLAB commands and algorithmic ideas requires a perhaps different skill that in some ways is more akin to learning a language. There is a tendency to do things in an inefficient way to begin with, but eventually one achieves fluency in most practical situations.

Before you get started on a project, it is a good idea to glance through the exercises to try to appreciate what is being asked. To answer most of the exercises you will have to find the relevant commands that enable MATLAB to do what you want. There are clues and guidance given for this within each project, although it will often be necessary (or at least helpful) to consult the MATLAB help system.

Each project is divided into several exercises, and earns a total of 20 marks. The projects must be completed to your satisfaction and submitted electronically before the respective deadlines in weeks 6 and 9, according to the instructions given below. The marks will count towards Prelims and will not be released until after the Preliminary Examinations.

Your answers will ideally display both your proficiency in MATLAB and appreciation of some of the underlying mathematics.

Each project has some marks set aside for "MATLAB code which is elegant and concise". The lecturer will (try to) give examples of such during the MATLAB lectures this term. As always, the presentation of your work also counts towards your grade.

1.3.1 Getting help

You may discuss with the demonstrators and others the techniques described in the Michaelmas Term Students' Guide, the commands listed in the Hilary Term Students' Guide, and those found in the MATLAB help pages. You may also ask the Head Demonstrator Quentin Parsons (`parsons@maths.ox.ac.uk`) or the Course Director to clarify any obscurity in the projects.

The projects must be your own unaided work. You will be asked to make a declaration to that effect when you submit them.

1.3.2 Debugging and correcting errors

'Debugging' means eliminating errors in the lines of code constituting a program. When you first devise a program for an exercise, do not be too disheartened if it does not work when you first try to run it. In that case, before attempting anything else, type `clear` at the command line and run it again. This has the effect of resetting all the variables, and may be successful at clearing the problem.

If the program still fails, locate the line where the problem originates. Remove semicolons from commands if necessary, so that intermediate calculations are printed out and you can spot the first line where things fail. You may also want to display additional output; the `disp()` command can be useful. If the program runs but gives the wrong answer, try running it for very simple cases, and find those for which it gives the wrong answer. Remove all code that is not used in that particular calculation, by inserting comments so that MATLAB ignores everything that follows on that line.

Website

A copy of this manual can be found at:

`https://courses.maths.ox.ac.uk/node/30`

This site will also incorporate up-to-date information on the course, such as corrections of any errors, possible hints on the exercises, and instructions for the submission of projects.

Legal stuff

Both the University of Oxford and the Mathematical Institute have rules governing the use of computers, and these should be consulted at `https://www.maths.ox.ac.uk/members/it/it-notices-policies/rules`.

Chapter 2

Preparing your project

To start, say, Project A, find the template 'projAtemplate.m' on the course website `https://courses.maths.ox.ac.uk/node/30` . Save this file as `projectA.m`, in a folder/directory also called `projectA`. Do not use other names.

You will be submitting this entire folder so please make sure it contains only files relevant to your project. You will almost certainly end up creating several `.m` files within this folder as part of your project.

2.1 Matlab publish

The file `projectA.m` should produce your complete answer. We will use the MATLAB 'publish' system.

```
publish projectA
```

This will create an HTML report in `projectA/html/projA.html` which can be viewed with a webbrowser (Firefox/Chrome/Safari/etc). The lecturer will give examples of 'publish' in your MATLAB lectures and post an example file on the course website. You should also read '`help publish`' and '`doc publish`'.

The examiners will read this published report in assessing your project. It is important that the report be well-presented.

- Divide `projectA.m` into headings for each exercise (perhaps more than one heading for each exercise).
- You can and should call other functions and scripts from within `projectA.m`
- Make sure you answer all the questions asked using text in comment blocks—if it asks why explain why!
- Some questions ask you to create a function in an external file. A good way to make this code appear in your published results is to include '`type other_function.m`' where appropriate in your `projectA.m`.
- Include appropriate MATLAB output: don't include pages and pages of output, but you must show that you have answered the question. This will require some thought and good judgment but it's worth the effort to avoid losing points if the examiners cannot determine your answer.

The examiners may also run your various codes and test your functions.

Make sure you run publish one last time before submitting your project. Then double-check the results.

2.2 Zip up your files

Make a `projectA.zip` or `projectA.tar.gz` file of your projectA folder or directory including all files and subfolders or subdirectories. No `.rar` files please. It is highly recommended you make sure you know how to do this well before the deadline.

Double-check that you have all files for your project and only those files for your project.

2.3 Submitting the projects

Full instructions on the submission system will be emailed to you nearer the time. The projects are to be submitted electronically at `https://courses.maths.ox.ac.uk/node/30/assignments` (from anywhere with internet access). Submission deadlines are given in Section 1.2. These deadlines are *strict*. It is *vital* that you meet them because the submission system will not allow submissions after the above times. You should therefore give yourself plenty of time to submit your projects, preferably at least a day or two in advance of the deadline. Penalties for late submission are specified in the Examination Conventions

`https://www.maths.ox.ac.uk/members/students/undergraduate-courses/examinations-assessments/examination-conventions`.

You will need your University Single Sign On username and password in order to submit each project, and also your examination candidate number (available from Student Self-Service). If you have forgotten your details you must contact OUCS well before the first deadline. The system will only allow one submission per project.

Chapter 3

Newton's Method and GPS (Project A)

3.1 Newton's Method

You will be familiar with the Newton-Raphson Method for numerically solving an equation $f(x) = 0$ in a single variable. Given a continuously differentiable function $f(x)$ defined for $x \in \mathbb{R}$ and a starting point x_0, the Newton-Raphson iteration is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{3.1}$$

Newton's Method usually refers to an extension of (3.1) to the case where we have n equations in n variables, where $n \geq 2$. If the equations are all linear, we are in the familiar realm of linear algebra, but here we are going to allow the equations to be nonlinear. Let's start simple: take $n = 2$ and suppose we have the two equations

$$f(x, y) = 0, \qquad g(x, y) = 0,$$

in the two real variables (x, y). Suppose our current point is (x_n, y_n). We can derive the Newton iteration by taking a linear approximation of the vector-valued function

$$\mathbf{F}(x, y) = \begin{bmatrix} f(x, y) \\ g(x, y) \end{bmatrix}$$

at (a, b), which takes the form

$$\mathbf{F}(x, y) = \begin{bmatrix} f(x, y) \\ g(x, y) \end{bmatrix} \approx \begin{bmatrix} f(x_n, y_n) \\ g(x_n, y_n) \end{bmatrix} + \begin{bmatrix} f_x(x_n, y_n) & f_y(x_n, y_n) \\ g_x(x_n, y_n) & g_y(x_n, y_n) \end{bmatrix} \begin{bmatrix} x - x_n \\ y - y_n \end{bmatrix}, \tag{3.2}$$

where $f_x(x_n, y_n)$ denotes the partial derivative of f with respect to x evaluated at the current point (x_n, y_n). The matrix

$$\mathbf{J}(x_n, y_n) = \begin{bmatrix} f_x(x_n, y_n) & f_y(x_n, y_n) \\ g_x(x_n, y_n) & g_y(x_n, y_n) \end{bmatrix}$$

is the Jacobian of $\mathbf{F}(x, y)$ evaluated at (x_n, y_n). We have now linearized the problem, and we can choose our next point (x_{n+1}, y_{n+1}) to be the one for which this approximation to $\mathbf{F}(x, y)$ is the zero vector. Setting (3.2) to zero and rearranging, we obtain the Newton iteration

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} - \begin{bmatrix} f_x(x_n, y_n) & f_y(x_n, y_n) \\ g_x(x_n, y_n) & g_y(x_n, y_n) \end{bmatrix}^{-1} \begin{bmatrix} f(x_n, y_n) \\ g(x_n, y_n) \end{bmatrix}. \tag{3.3}$$

Notice the conceptual similarity to the 1D Newton-Raphson iteration!

Suppose we wish to find the points of intersection of the curves $x^4 + y^4 = 10$ and $(x - 2)^2 + (y - 1)^2 = 3$. This is two equations in two unknowns, and we can write it in the form above as

$$\mathbf{F}(x,y) = \begin{bmatrix} f_1(x,y) \\ f_2(x,y) \end{bmatrix} = \begin{bmatrix} x^4 + y^4 - 10 \\ (x-2)^2 + (y-1)^2 - 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{3.4}$$

Exercise 3.1. *Use either the* `fimplicit` *or* `ezplot` *function to plot graphs of $f_1(x,y)$ and $f_2(x,y)$ on the same figure. Use different colours for the two curves and label the graphs.*

Let's solve the system of equations (3.4) using Newton's Method. In this case, the Jacobian $\mathbf{J}(x,y)$ is

$$\mathbf{J}(x,y) = \begin{bmatrix} 4x^3 & 4y^3 \\ 2(x-2) & 2(y-1) \end{bmatrix}.$$

Exercise 3.2. *Let $\mathbf{F}(x,y)$ be as given in (3.4). Write a function* `evaluate1.m` *which takes as input the coordinates (x,y) (either as a vector or as separate arguments) and outputs the length-2 vector $\mathbf{F}(x,y)$ and the 2×2 Jacobian matrix $\mathbf{J}(x,y)$ evaluated at the point (x,y).*

Exercise 3.3. *Suppose we solve (3.4) numerically using Newton's Method starting at the point $(x_0, y_0) = (4, -2)$. Using your* `evaluate1.m` *function, show that the next Newton iterate is $(x_1, y_1) \approx (3.1108, -0.9261)$. Use a* `for` *loop to extend your code to perform the first ten iterations of Newton's Method, and display the iterates $(x_1, y_1), (x_2, y_2), \ldots, (x_{10}, y_{10})$.*

You should find that Newton's Method converges to one of the points of intersection of the two curves.

Exercise 3.4. *Use Newton's Method to find the other point of intersection.*

3.2 Beyond two dimensions

You can probably guess how Newton's Method extends to solving m nonlinear equations in m unknowns, where $m > 2$. Consider the m-dimensional function of m variables

$$\mathbf{F}(x_1, x_2, \ldots, x_m) = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_m) \\ f_2(x_1, x_2, \ldots, x_m) \\ \vdots \\ f_m(x_1, x_2, \ldots, x_m) \end{bmatrix}.$$

Its Jacobian $\mathbf{J}(x_1, x_2, \ldots, x_m)$ is the $m \times m$ matrix

$$\mathbf{J}(x_1, x_2, \ldots, x_m) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_m} \end{bmatrix},$$

where each of the partial derivatives is evaluated at the point $(x_1, x_2, \ldots, x_m)$. Writing

$$\mathbf{x^n} = \begin{bmatrix} x_1^n \\ x_2^n \\ \vdots \\ x_m^n \end{bmatrix}$$

for the vector of values of the variables $(x_1, x_2, \ldots, x_m)$ at iteration n (note the use of the superscript this time for the iteration number), the Newton iteration is

$$\mathbf{x^{n+1}} = \mathbf{x^n} - \mathbf{J}^{-1}\mathbf{F},$$

where both $\mathbf{J}$ and $\mathbf{F}$ are evaluated at the current point $(x_1^n, x_2^n, \ldots, x_m^n)$.

Let's use Newton's Method to numerically find a solution to the equations

$$\begin{cases} 15x_1 + x_2^2 - 4x_3 &= 13, \\ x_2^2 + 10x_2 - e^{-x_3} &= 11, \\ x_2^3 - 25x_3 &= -22, \end{cases}$$

which we may formulate at the following system of equations.

$$\mathbf{F}(x_1, x_2, x_3) = \begin{bmatrix} f_1(x_1,x_2,x_3) \\ f_2(x_1,x_2,x_3) \\ f_3(x_1,x_2,x_3) \end{bmatrix} = \begin{bmatrix} 15x_1 + x_2^2 - 4x_3 - 13 \\ x_2^2 + 10x_2 - e^{-x_3} - 11 \\ x_2^3 - 25x_3 + 22 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{3.5}$$

Exercise 3.5. *Repeat Exercise 3.2 for the system of equations (3.5). Write a function* `evaluate2.m` *which takes as input the coordinates (x_1, x_2, x_3) (either as a vector or as separate arguments) and outputs the length-3 vector $\mathbf{F}(x_1, x_2, x_3)$ and the 3×3 Jacobian matrix $\mathbf{J}(x_1, x_2, x_3)$ evaluated at the point (x_1, x_2, x_3).*

Exercise 3.6. *Using your* `evaluate2.m` *function, apply Newton's Method to numerically find a solution to the system of equations (3.5). Make sure to use sufficiently many iterations to achieve convergence. Use any starting point you like. Display your answer and verify by numerical calculation that it satisfies the original equations.*

3.3 The GPS problem

The Global Positioning System (GPS) provides geolocation information to a receiver by interpreting the information in signals from several satellites. What follows is a somewhat simplified description of how it works, but which captures the main idea.

Suppose that signals are sent to a receiver from four satellites, giving their position and the time at which the signal was sent. The receiver has its own clock and, for each satellite, records the difference Δt between the time the signal was sent and the time it was received. Assuming the signal travels at the speed of light, the distance to the satellite is therefore $c\Delta t$. However, the receiver's clock is unlikely to be perfectly synchronized with the satellite's atomic clock — even a millisecond error would lead to an unacceptable distance error of several hundred kilometres — so it is better to take the distance to be $c(\Delta t - \epsilon)$ for some unknown clock error ϵ.

Let's use a Cartesian xyz-coordinate system with origin at the centre of the earth, and the unit of measure being the radius of the earth, as depicted in Figure 3.1.

Let's measure time in milliseconds (one thousandth of a second), so that the speed of light is $c \approx 0.04706$ radii per millisecond. Suppose a satellite gives its position as being (X, Y, Z), and the receiver calculates its time difference to be Δt. This information can be expressed as the following equation for (x, y, z), the position of the receiver, and ϵ, the clock error.

$$(x - X)^2 + (y - Y)^2 + (z - Z)^2 = c^2(\Delta t - \epsilon)^2. \tag{3.6}$$

Note that (X, Y, Z), Δt and c are all constants. By forming an equation of this form using the information from each of the four satellites, we arrive at a system of four equations in four unknowns.

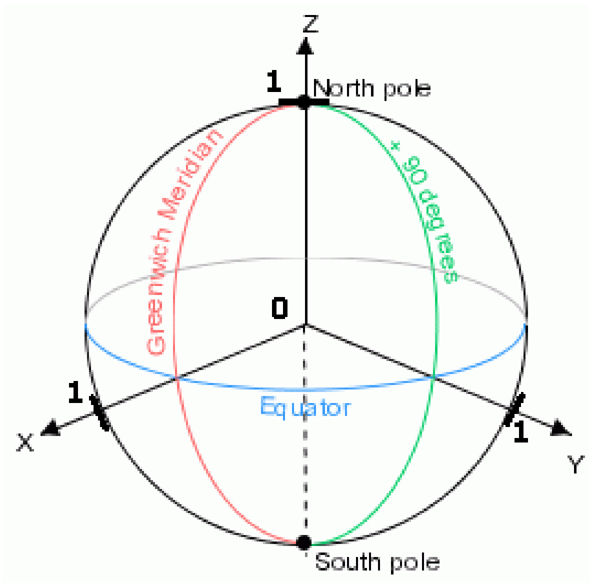


Figure 3.1: The Cartesian coordinate system we are using.

Let's consider a specific problem. The information obtained from four satellites is summarized in Table 3.1. This information leads to a system of four equations in the four unknowns

Satellite	(X, Y, Z)	Δt
1	$(1, 2, 0)$	40.31
2	$(2, 0, 2)$	63.59
3	$(1, 1, 1)$	30.92
4	$(2, 1, 0)$	50.33

Table 3.1: Information from four satellites.

x, y, z and ϵ.

Exercise 3.7. *Use the measurements in Table 3.1, together with equation (3.6), to deduce a system of four equations in the four variables x, y, z and ϵ. Apply Newton's Method to numerically find two solutions to your system of equations. You may take $c^2 = 2.214 \times 10^{-3}$. Explain which of the two solutions you obtain is likely to correspond to a receiver on the surface of the earth. State clearly your solution for both the position of the receiver (x, y, z) and the clock error ϵ. Hint: You may find it helpful to define a suitable vector function $F(x, y, z, \epsilon)$.*

We don't tend to think about locations on the surface of the earth in terms of Cartesian co-ordinates. More likely, we would think in terms of longitude (degrees East of the Greenwich Meridian, with negative values indicating a location West of the Greenwich Meridian), latitude (degrees North of the equator, with negative values indicating a location South of the equator) and altitude (height above the surface of the earth, say in m). Longitude and latitude are displayed as λ and ϕ respectively in Figure 3.2.

Let's assume that the Cartesian co-ordinates we have defined are such that the positive z-axis points North, the plane containing the Greenwich Meridian is the plane containing the x and z axes, just as originally shown in Figure 3.1.

Exercise 3.8. *Assuming that the Earth is a perfect sphere of radius* 1*, calculate the longitude and latitude of the receiver (both in degrees), and the altitude of the receiver in m. You can*

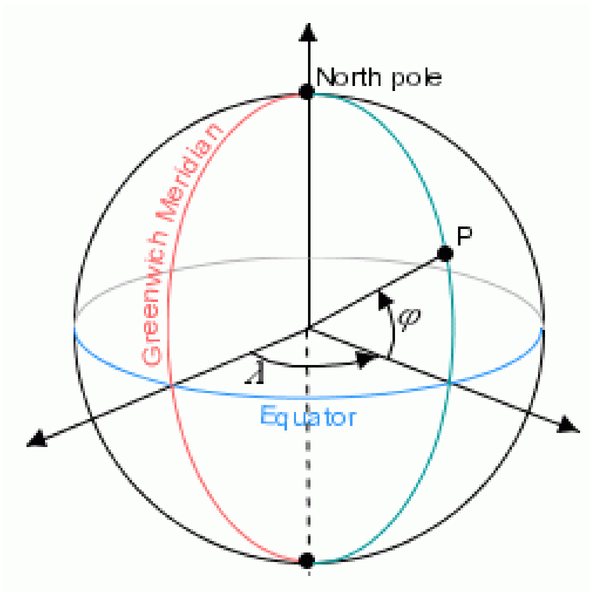


Figure 3.2: Longitude and latitude.

assume that the radius of the Earth is $6371.008km$. *Hint: you will need to convert the result from Exercise 3.7 into spherical polar co-ordinates.*

You might like to drop the longitude and latitude figures you obtained into Google Maps and reflect on how realistic is the altitude you obtain!

3.4 Postscript

While the previous section illustrates the fundamentals of the mathematics of GPS, a large number of simplifying assumptions have been made. While the speed of light in a vacuum is constant, radio waves are slowed down a fraction by the earth's atmosphere, and by taking indirect paths (for example being reflected off buildings), and receivers need to correct for these effects. Indeed, our somewhat cavalier approximation of the speed of light to a handful of significant figures would be frowned upon in practice!

Furthermore, the earth is not a perfect sphere. The World Geodetic System (WGS) used by GPS treats the surface of the earth as a spheroid whose equatorial radius is greater than its polar radius. The conversion from Cartesian to spheroidal coordinates then becomes less straightforward. An interesting investigation would be to repeat Exercise 3.8 and calculate latitude, longitude and altitude with respect to more realistic spheroidal coordinates.

While the numerical solution of the GPS location problem is very much about solving nonlinear systems of equations, in practice it is likely to take a more sophisticated form to the one presented in this project. More than four satellite measurements may be available, which leads to a 'best fit' least-squares problem, rather than an exact solution problem. Accuracy is also usually improved by repeating measurements and analyzing the combined data. To achieve the remarkable accuracy that we come to expect today, the technique of *differential GPS* is used in which two receivers are used: one known and one unknown. Those interested in exploring more are referred to *Linear Algebra, Geodesy and GPS* by Gilbert Strang and Kai Borre (Wellesley-Cambridge 1997) for further reading.

Chapter 4

Prony's Method (Project B)

4.1 Weighted sums of complex exponentials

This project is about functions which are weighted sums of complex exponentials, that is, expressions of the form

$$f(t) := \sum_{j=1}^{k} a_j e^{i\theta_j t},$$

where k is a positive integer and where we will assume $\theta_j \in (-\pi, \pi]$. Such a function is uniquely defined by specifying k along with the length-k vectors

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_k \end{bmatrix}, \quad a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix}.$$

We will be working with complex numbers in this project, which is on the whole no different in MATLAB to working with real numbers. One possible snag to flag up in advance: if `A` is a complex-valued matrix, `A'` is its conjugate transpose, that is, the transpose with the complex conjugate taken of each entry. The same is true for vectors. To perform the transpose of a matrix/vector (without taking complex conjugates), you can use `transpose(A)`.

Exercise 4.1. *Consider the particular instance of*

$$f(t) = 3e^{0.5it} + 4e^{-1.5it} + 5e^{2.3it} + 6e^{0.8it},$$

which corresponds to taking

$$\theta = \begin{bmatrix} 0.5 \\ -1.5 \\ 2.3 \\ 0.8 \end{bmatrix}, \quad a = \begin{bmatrix} 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}.$$

The function $f(t)$ is complex-valued, but we can visualise it by plotting its absolute value. Plot a graph of $|f(t)|$ over the range $t = [0, 12]$ and check visually that it agrees with the one in Figure 4.1.

Exercise 4.2. *With $f(t)$ defined as in Exercise 4.1, plot Re$[f(t)]$ and Im$[f(t)]$ in the same figure. Use different colours for each and use a legend to label the plots.*

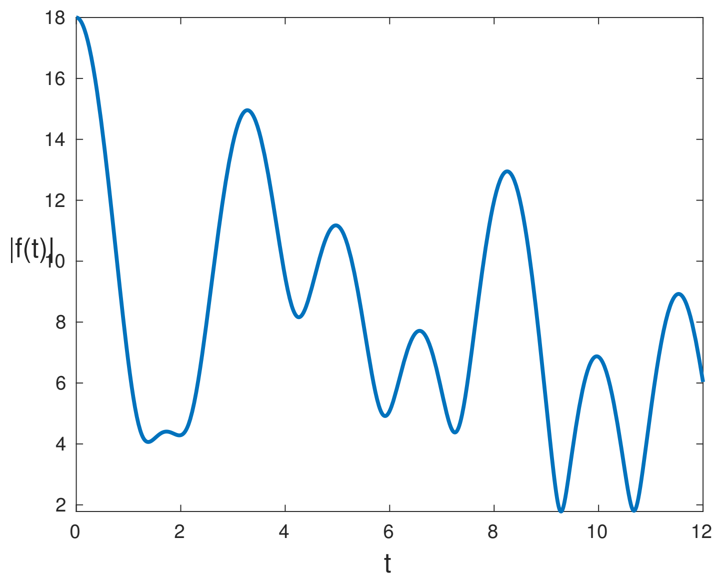


Figure 4.1: A plot of $|f(t)|$ as given in Exercise 4.1.

4.2 Calculating the exponents from samples

Suppose in general that we know that $f(t)$ is a weighted sum of k complex exponentials, but we do not know what its coefficients θ and a are. Suppose that we are able to *sample* $f(t)$ at the discrete, equally-spaced set of points $\{1, 2, \ldots, 2k\}$. In other words, we have the $2k$ (complex) function values $f(1)$, $f(2)$, up to $f(2k)$. Let us write these function values as $f_1, f_2, \ldots, f_{2k}$, and write

$$f := \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{2k} \end{bmatrix}$$

for the vector whose entries are these function values. In the case of the function defined in Exercise 4.1, we have $k = 4$ complex exponentials, and we take $2k = 8$ samples at $t = 1, 2, \ldots, 8$.

Exercise 4.3. *Compute the vector f of samples $f_1, f_2, \ldots, f_8$ for the function defined in Exercise 4.1 and display the vector. Hint: you will need to convert from symbolic variables to doubles.*

Prony's Method, devised in 1795 by the French mathematician Gaspard Riche de Prony, is an algorithm for calculating the coefficients θ from the samples f. The method is as follows.

1. Form the $k \times k$ Hankel matrix[1] H and the length-k vector g, defined by

$$H := \begin{bmatrix} f_1 & f_2 & f_3 & \cdots & \cdots & f_{k-1} & f_k \\ f_2 & f_3 & & & \iddots & \iddots & f_{k+1} \\ f_3 & & & \iddots & \iddots & \iddots & \vdots \\ \vdots & & \iddots & \iddots & \iddots & & \vdots \\ \vdots & \iddots & \iddots & \iddots & & & f_{2k-3} \\ f_{k-1} & \iddots & \iddots & & & f_{2k-3} & f_{2k-2} \\ f_k & f_{k+1} & \cdots & \cdots & f_{2k-3} & f_{2k-2} & f_{2k-1} \end{bmatrix}, \quad g := \begin{bmatrix} f_{k+1} \\ f_{k+2} \\ \vdots \\ f_{2k} \end{bmatrix}$$

   The matrix H can be easily made in MATLAB with the help of the `hankel` command.

2. Solve the system of linear equations $H\lambda = g$, to give

$$\lambda = \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{k-1} \end{bmatrix}.$$

   You can use the backslash command `H\g` to solve the linear system.

3. Find the k (complex) roots, $r_1, r_2, \ldots, r_k$ of the polynomial

$$z^k - \lambda_{k-1}z^{k-1} - \lambda_{k-2}z^{k-2} - \ldots - \lambda_1 z - \lambda_0 = 0.$$

   You might find the `roots` command useful.

4. Compute

$$\theta_j = \operatorname{Im}(\operatorname{Log} r_j), \quad j = 1, 2, \ldots, k,$$

   where Log z denotes the *principal logarithm*[2] of the complex number z. The full vector θ can be computed in one go in MATLAB using `theta=imag(log(r));`.

Exercise 4.4. *Implement Prony's Method in* MATLAB *on the samples you computed in Exercise 4.3. Check that you obtain the correct values for θ.*

4.3 Calculating the amplitudes

Now we have the correct exponents $\{\theta_j\}$, obtaining their corresponding amplitudes $\{a_j\}$ amounts to solving an overdetermined linear system. We have

$$f_m = \sum_{n=1}^{k} a_n e^{i\theta_n m}$$

for all $m = 1, 2, \ldots, 2k$, which can be written as the matrix equation

$$f = Ma, \tag{4.1}$$

where M is the $2k \times k$ matrix whose entries are

$$M_{mn} = e^{i\theta_n m}.$$

Exercise 4.5. *Construct and solve the linear system (4.1) using the samples you computed in Exercise 4.3 (and the correct amplitudes). Check that you obtain the correct values for a.*

[1]A Hankel matrix is one in which each skew-diagonal is constant.

[2]The imaginary part of the principal logarithm of a complex number $x+iy$ is also equal to the four-quadrant inverse tangent of y/x, which can be calculated in MATLAB using `atan2(y,x)`.

4.4 An unknown waveform to decipher

A function $f(t)$ is known to consist of a sum of five sinusoids, but their exponents and amplitudes are unknown. The only information you have available is ten samples, taken at the values $t = 1, 2, \dots, 10$. The samples are given in Table 4.1.

t	1	2	3	4	5	6	7	8	9	10
$f(t)$	6.3185	-9.4712	-3.1924	-4.7683	4.7884	24.0638	8.4936	-13.0161	-4.6636	0.3716

Table 4.1: Ten samples of an unknown waveform.

Notice that this time all the samples are real-valued. But not to worry, a real number is just a complex number with no imaginary part and so we may still try to decipher the waveform using Prony's Method. That said, you might like to ponder what kind of a sum of complex exponentials would give rise to real-valued samples!

Exercise 4.6. *Use Prony's Method to find the exponents and amplitudes of a sum of five complex exponentials which fit the samples given in Table 4.3. Using the Symbolic Math Toolbox or pen-and-paper calculation, give a simplified expression for $f(t)$. Plot a graph of $f(t)$ for $t \in [-15, 15]$. Using $\times$ markers and a different colour, plot the values of the samples on the same graph and check that they agree.*

4.5 Extension to decaying exponentials

In practice, naturally occurring oscillations are likely to be damped, exhibiting exponential decay. In this case, we probably want to consider a different model, namely

$$f(t) := \sum_{j=1}^{k} a_j e^{(\sigma_j + i\theta_j)t},$$

where k is a positive integer and $\theta_j \in (-\pi, \pi]$ as before for all j, and where the $\{\sigma_j\}$ are real numbers. If $\sigma_j > 0$ we have exponential growth, if $\sigma_j < 0$ we have exponential decay, and if $\sigma_j = 0$ we have an undamped complex exponential as before.

Two small change are needed to Prony's Method

- In the final step of the calculation of the exponents, calculate σ_j as the real part of the principal logarithm of r_j (see Section 4.2), that is,

$$\sigma_j = \mathrm{Re}(\mathrm{Log}\, r_j), \quad j = 1, 2, \dots, k.$$

- In the calculation of the amplitudes, we now have

$$f_m = \sum_{n=1}^{k} a_n e^{(\sigma_n + i\theta_n)m}$$

for all $m = 1, 2, \dots, 2k$, and so the entries of the matrix M become

$$M_{mn} = e^{(\sigma_n + i\theta_n)m}.$$

Now for a final problem. It is given that $f(t)$ is a single decaying cosine function of the form

$$f(t) = ae^{\sigma t} \cos(\theta t + \phi),$$

where $a > 0$, $\sigma < 0$, $\theta \in (-\pi, \pi]$ and $\phi \in (-\pi, \pi]$ are unknown. Here ϕ is a phase shift. You are also given the samples

t	1	2	3	4
$f(t)$	0.7153	-0.3497	-0.3861	-0.0831

Exercise 4.7. *Find a, σ, θ and ϕ and plot $f(t)$ for $t \in [0, 8]$.*

4.6 Postscript

Two questions may have arisen in your mind: why does Prony's Method work, and what is it useful for?

Take the first, and let's consider the simplest case in which we are recovering a sum of k undamped complex exponentials

$$f(t) := \sum_{j=1}^{k} a_j e^{i\theta_j t} \tag{4.2}$$

from the $2k$ samples $f_1 = f(1)$, $f_2 = f(2)$, up to $f_{2k} = f(2k)$. The method revolves around finding the coefficients $\lambda_0, \lambda_1, \ldots, \lambda_{k-1}$ of a polynomial

$$p(z) := z^k - \lambda_{k-1} z^{k-1} - \lambda_{k-2} z^{k-2} - \ldots - \lambda_1 z - \lambda_0 = 0 \tag{4.3}$$

in z whose roots are $e^{i\theta_j}$ for $j = 1, 2, \ldots, k$. Suppose these are the roots. Then $e^{i\theta_j} p(e^{i\theta_j}) = 0$ for all $j = 1, 2, \ldots, k$, and we have

$$\sum_{j=1}^{k} a_j \left[e^{i(k+1)\theta_j} - \lambda_{k-1} e^{ik\theta_j} - \ldots - \lambda_0 e^{i\theta_j} \right] = 0. \tag{4.4}$$

Making appropriate substitutions of (4.2) into (4.4), we then deduce

$$y_{k+1} - \lambda_{k-1} y_k - \ldots - \lambda_0 y_1 = 0.$$

We can obtain similar linear equations in λ by using $e^{il\theta_j} p(e^{i\theta_j}) = 0$ for each $l = 2, 3, \ldots, k$, finally arriving at

$$y_{k+l} - \lambda_{k-1} y_{k+l-1} - \ldots - \lambda_0 y_l = 0, \quad l = 1, 2 \ldots, k.$$

This is a system of k equations in k unknowns, and is precisely the one defined by the Hankel matrix equation $H\lambda = g$. A similar argument can be followed in the case of damped exponentials.

As far as the usefulness of Prony's Method is concerned, there are many practical applications in frequency detection/estimation which amount to decomposing a signal into a sum of sinusoids. Examples include the analysis of radio signals in communications, and acoustic signals of various kinds (including music). One popular tool for analysing the frequency composition of a signal is the Discrete Fourier Transform (DFT), which can be thought of as a discrete kind of Fourier analysis. This approach requires the assumption that the frequencies take values on some discrete grid. Prony's Method, on the other hand, makes no such assumption, allowing frequencies to be estimated with greater precision. Popular frequency estimation algorithms based upon Prony's Method include the MUSIC algorithm, a variant of which is included in the MATLAB Signal Processing Toolbox.

Chapter 5

Interpolation and the Cayley-Bacharach Theorem (Project C)

5.1 Basic Python and Sage

For this project, we will use Sage through the SageMathCloud:

`http://cloud.sagemath.org`

You should be able to use this from any computer with a recent web browser. If you prefer and you use a GNU/Linux or Mac system, you can download Sage onto your own machine from

`http://sagemath.org`.

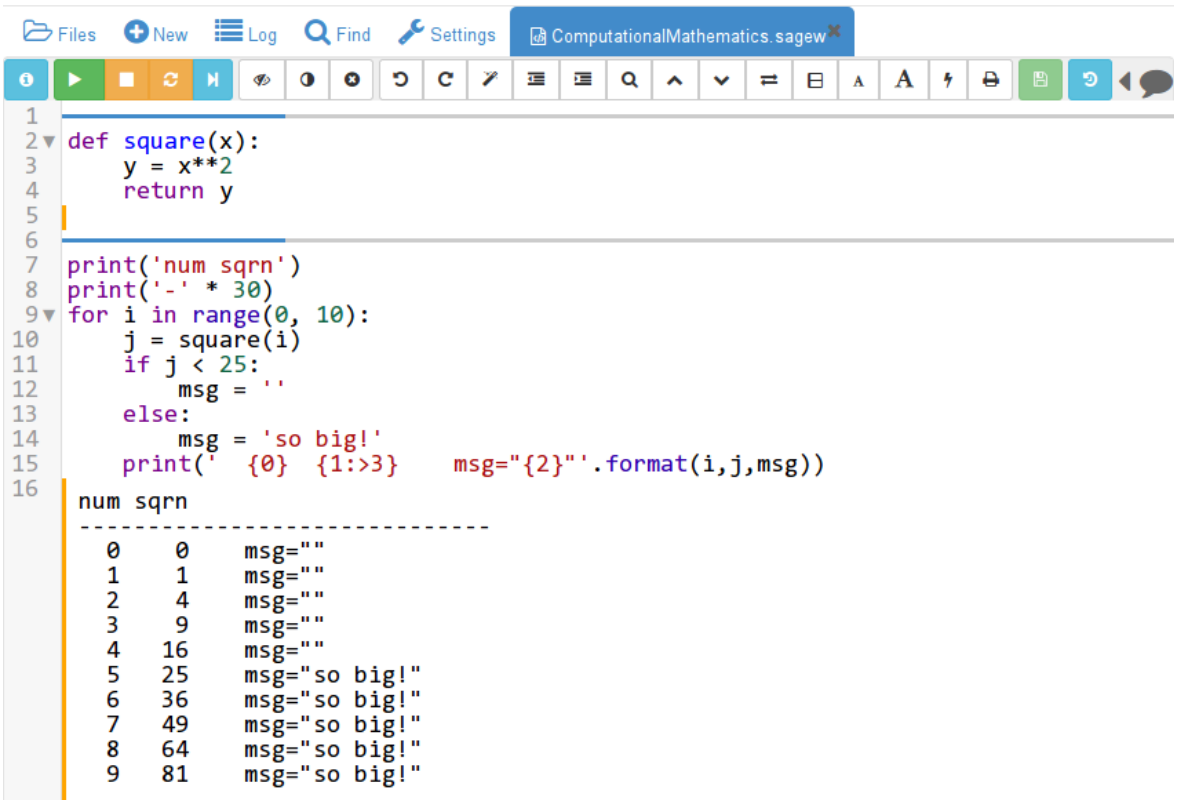
Warning: Sage is *not* MATLAB : you'll need to learn some aspects of the Python programming language to complete this project. The course demonstrators may or may not be able to help. To compensate, the exercises here should be a bit easier.

5.1.1 Getting started

1. Create an account on `http://cloud.sagemath.org` via the email invitation.
2. You will then be given access to the Computational Mathematics Course SageMathCloud project.
3. Open the "Assignment" worksheet.
4. Put your answers into the worksheet.
5. To evaluate in the worksheet, press shift-enter. (For example, to evaluate 1+1, type "1 + 1" into a worksheet, press "shift" and "enter" and verify that you get 2.)
6. See Section 5.6 for notes on submitting your Sage project.

Sage is written in Python and uses the Python language, with a few modifications to make it more suitable for mathematics as its input. So let's start with some programming and flow control basics in Python. These are somewhat similar to many other programming languages, including MATLAB . Sage (and Python) have lots of online documentation[1]. And if none

[1]`http://www.sagemath.org/help.html#SageStandardDoc`



```
def square(x):
    y = x**2
    return y

print('num sqrn')
print('-' * 30)
for i in range(0, 10):
    j = square(i)
    if j < 25:
        msg = ''
    else:
        msg = 'so big!'
    print('  {0}  {1:>3}    msg="{2}"'.format(i,j,msg))
```

```
num sqrn
------------------------------
  0    0    msg=""
  1    1    msg=""
  2    4    msg=""
  3    9    msg=""
  4   16    msg=""
  5   25    msg="so big!"
  6   36    msg="so big!"
  7   49    msg="so big!"
  8   64    msg="so big!"
  9   81    msg="so big!"
```

Figure 5.1: Screenshot of the SageMathCloud user interface.

of that answers your question, consider asking on the *Sage-support mailinglist*[2] or the *Sage askbot*[3].

Probably the most important distinction is that whitespace (indenting) is important in Python whereas it is mostly cosmetic in MATLAB . Here is an example of defining a function (procedure) in Python:[4]

```
sage: def square(x):
sage:     y = x^2
sage:     return y
sage: square(7)
49
```

Figure 5.1 includes an example of "for loop" and "if-then-else" flow control in Python. Careful with the whitespace!

Python is an object-oriented programming language, and all mathematical objects in Sage are objects of some kind. At the most basic level that means that most functionality is accessed as `x.compute()` instead of `compute(x)` for any given variable `x`. For example[5]

```
sage: D8 = DihedralGroup(8)
sage: D8.center()     # instead of center(D8)
Subgroup of (Dihedral group of order 16 as a permutation group)
    generated by [(1,5)(2,6)(3,7)(4,8)]
```

To get help about anything in Sage just add a question mark at the end:

[2]https://groups.google.com/forum/#!forum/sage-support

[3]http://ask.sagemath.org

[4]`sage:` is the input prompt if you use the command-line interface to Sage. It might look different in other user interfaces; for example, there is no prompt on the input cell in SageMathCloud. You should only type in what comes after the `sage:` prompt.

[5]The dihedral group are the symmetries of a polygon.

```
sage: D8?            # help about DihedralGroup
sage: D8.center?     # help about the center method
```

The advantage of this "reversed" notation is that tab-completion makes it easy to explore: just press the `Tab` key to get a context-sensitive list of completions.

```
sage: D8.<tab>       # press tab key after the period, prints all methods
sage: D8.is_s<tab>   # list methods starting with ``is_s''
D8.is_semi_regular    D8.is_solvable       D8.is_supersolvable
D8.is_simple          D8.is_subgroup
```

5.2 Rings and polynomials

Now for some mathematics, which we'll explore using Sage. A **ring** is a structure like the integers $\mathbb{Z}$ where you have two operations (addition and multiplication) that behave in the "familiar" way[6]. The ring of integers $\mathbb{Z}$ can be created in Sage using the command

```
sage: ZZ
```

The real numbers $\mathbb{R}$, the complex numbers $\mathbb{C}$ and the rational numbers $\mathbb{Q}$ — each with the usual operations $(+, \times)$ — are also rings. Here's how to create the ring of rational nuumbers $\mathbb{Q}$ in Sage.

```
sage: QQ      # rational numbers
```

A ring which satisfies an additional axiom (namely the existence of multiplicative inverses) is called a **field**. Note that $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$ are fields, but $\mathbb{Z}$ is not.

Polynomials are of course familiar territory, but it will help to have a precise definition. A *polynomial* in *variables* $x_1, \dots, x_n$ over a ring R is a finite sum

$$f(x_1, \dots, x_n) = \sum_{I:=(i_1,\dots,i_n)\geq 0} a_I x^I, \qquad a_I \in R,\ x^I := x_1^{i_1} x_2^{i_2} \dots x_n^{i_n}. \tag{5.1}$$

The x^I in (5.1) are referred to as the *monomials*, and the a_I are their corresponding *coefficients*. The (finite) set of I for which $a_I \neq 0$ is called the *support* of f.

The polynomials in $x_1, \dots, x_n$ over R form a ring $R[x_1, \dots, x_n]$, with addition and multiplication defined as in the univariate case. The *evaluation* of f at $(t_1, \dots, t_n) \in R^n$ is $f(t_1, \dots t_n) \in R$. The *degree* of f is the maximum of $|I| := \sum_{j=1}^n i_j$ as I ranges over the support of f. We say that f is *homogeneous* if all its monomials have the same degree.

For example,

$$f(x, y) = x^2y - 2y^2$$

has monomials x^2y and y^2 and $I = \{(2, 1), (0, 2)\}$, and has degree 3. It is not homogeneous because the monomials do not all have degree 3.

In this project we will explore the behaviour of polynomials over a *field* $\mathbb{K}$, such as $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$. However, in all the examples and exercises we will take $\mathbb{K} := \mathbb{Q}$, the rationals. We can create a bivariate polynomial ring $\mathbb{Q}[x, y]$ in Sage using the following command.

```
sage: R.<x,y> = QQ[]
```

And here is how to compute evaluations of a polynomial, compute its degree and factorize it.

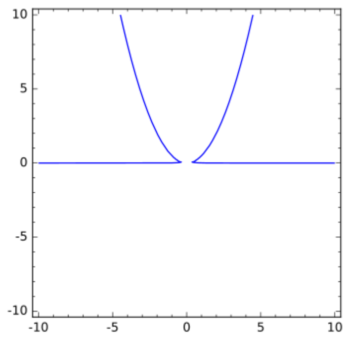
```
sage: R.<x,y> = QQ[]
sage: f = x^2*y-2*y^2
sage: f in R # indeed, f is an element of R
```

[6]You will meet the notion of **groups** this term in the *Groups and Group Actions* course. A ring is essentially a group with a second operation ·, and for which additional axioms hold.

```
True
sage: f(1,2)
-6
sage: f(-1,2/3)
-2/9
sage: f.degree()
3
sage: f.monomials()
[x^2 y, y^2]
sage: f.coefficients()
[1, -2]
sage: f.factor()
(-1) * y * (-x^2 + 2*y)
```

Given $g \in \mathbb{K}[x, y]$, that is, a bivariate polynomial over a field $\mathbb{K}$, the set of zeros of g is usually referred to as the *plane curve* defined by g, and it is denoted by $g = 0$. To plot curves in Sage, one can use the implicit plot facility, which takes Sage *functions*[7] as input, as follows.

```
sage: pf(a,b)=f(a,b)/10    # scale down for more detail
sage: implicit_plot(pf==0,(-10,10),(-10,10)) # displays plot
```



5.3 Projective curves

Given a non-homogeneous bivariate polynomial $g \in \mathbb{K}[x, y]$, we can obtain a related homogeneous polynomial $\tilde{g}$ by introducing a third variable z. If g has degree d, then the *homogenisation* $\tilde{g}$ of g is defined to be

$$\tilde{g}(x, y, z) := z^d g(x/z, y/z).$$

For instance, if $g(x, y) = 5x^2 - xy + 1$, we have $d = 2$, and its homogenisation $\tilde{g}(x, y, z)$ is

$$\tilde{g}(x, y, z) = z^2[5(x/z)^2 - (x/z)(y/z) + 1] = 5x^2 - xy + z^2.$$

Consider the set of zeros of $\tilde{g}$. We see, for example, that $(1, 5, 0)$ is a zero of $\tilde{g}$, but also so is any scalar multiple $\lambda(1, 5, 0) = (\lambda, 5\lambda, 0)$. This is always true for homogeneous polynomials: their zeros consist of lines through the origin. We now choose to view these lines through the origin as points in *projective space*. For example, we view the line $\lambda(1, 5, 0)$ as the *projective point* $(1 : 5 : 0)$. The *projective plane* is the set of all $(a : b : c)$ for which a, b and c are not all zero. Note that projective points are unique up to scalar multiplication, so $(1 : 5 : 0)$ is the same as $(2 : 10 : 0)$ for example.

Given a bivariate polynomial g, the set of points for which $g = 0$ (its plane curve) corresponds to some *projective curve*, which is the set of projective points for which $\tilde{g} = 0$. It turns out to be advantageous to consider the projective situation, and to study the projective curves defined by homogeneous polynomials[8].

[7]Note that, in the above example, Sage views `f` as a polynomial and not as a function, whereas `pf` is a function built from the polynomial.

[8]For a more rigorous treatment, see the Projective Geometry course!

5.4 Interpolation with polynomials

By polynomial interpolation we mean finding a polynomial that passes through a number of given points. A fundamental result about interpolation is that a polynomial of degree d is uniquely determined by its evaluations at $d+1$ points. In what follows it will be convenient to denote the set of degree d homogeneous polynomials by $\mathbb{K}_d[x,y,z]$.

The simplest kind of homogeneous polynomials are linear ones, in $\mathbb{K}_1[x,y,z]$; they define projective lines. In particular, recall the formula for the equation $f(x) = 0$ of a line through two points from the Geometry course. The same formula also applies in projective space: the equation of the line passing through two distinct projective points $A = (\alpha_1 : \alpha_2 : \alpha_3)$ and $B = (\beta_1 : \beta_2 : \beta_3)$ is

$$\det \begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \\ x & y & z \end{pmatrix} = 0 \tag{5.2}$$

If we fix a point $X = (x : y : z)$, one may also interpret (5.2) as the condition for the points A, B and X to be collinear.

In Sage one may define matrices over polynomial rings, e.g. the following computes f as a determinant of a matrix, so that $f = 0$ is the equation of the line through the origin and the point $(1,2)$.

```
sage: R.<x,y>=QQ[]
sage: M=matrix([[1,2],[x,y]]); M
[ 1  2]
[ x  y]
sage: f=M.det(); f      # the determinant of M
-2*x + y
sage: f(8,-77/3)        # evaluate f fully
-125/3
sage: f(x=8)           # or partially
x2 - 16
```

Exercise 5.1. *Compute an (explicit) equation of the line L through the points $(1 : 2 : 3)$ and $(0 : 1 : -1)$ using Sage. Use it to check whether $(1 : 3 : 2)$ lies on L.*

So for $d = 1$ we observe that 2 points uniquely determines a line, as expected. A similar phenomenon happens with *projective conics*, i.e. projective curves defined by polynomials of degree 2. Namely, 5 points, *with no 4 of them collinear*, determine a conic uniquely, and there is a formula for the equation of the conic in terms of coordinates of these points. Such a formula is given by (5.2). Specifically, let $P_j = (x_j : y_j : z_j)$, $1 \leq j \leq 5$.

$$M(x,y,z) := \begin{pmatrix} x_1^2 & y_1^2 & z_1^2 & x_1y_1 & x_1z_1 & y_1z_1 \\ x_2^2 & y_2^2 & z_2^2 & x_2y_2 & x_2z_2 & y_2z_2 \\ x_3^2 & y_3^2 & z_3^2 & x_3y_3 & x_3z_3 & y_3z_3 \\ x_4^2 & y_4^2 & z_4^2 & x_4y_4 & x_4z_4 & y_4z_4 \\ x_5^2 & y_5^2 & z_5^2 & x_5y_5 & x_5z_5 & y_5z_5 \\ x^2 & y^2 & z^2 & xy & xz & yz \end{pmatrix}, \qquad f(x,y,z) := \det M(x,y,z) \tag{5.3}$$

Naturally $f(x', y', z') = 0$ if the 6 points $P_1, \dots, P_5$ and $P_6 := (x' : y' : z')$ lie on a conic. The rows of the somewhat puzzlingly looking matrix in (5.3) are merely coefficients of the monomials of degree 2 in 3 variables evaluated at the points $P_1, \dots, P_6$, and the vanishing of the determinant means that the condition imposed by P_6 on the coefficients of polynomials in $\mathbb{K}_2[x,y,z]$ is a linear combination of the conditions imposed by the other points.

Moreover, the rank of $M(x', y', z')$ is exactly 5 — unless the condition that no 4 of $P_1, \dots, P_5$ are collinear is violated.

The following code defines a function which generates the matrix $M(x, y, z)$ given a set of points $P_1, \dots, P_5$.

```
sage: R.<x,y,z>=QQ[]
sage: v2=vector([x^2,y^2,z^2,x*y,x*z,y*z])
sage: def M(P): return matrix(map(lambda p: v2(p[0],p[1],p[2]), P)+[v2])
```

Now let's use Sage to illustrate what happens if we take the points

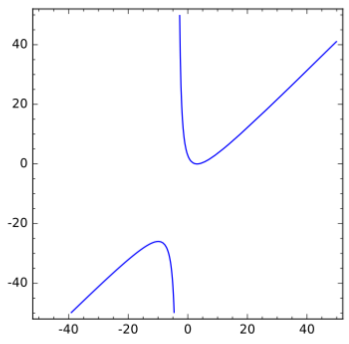
$$P_1 = (0:0:1), \quad P_2 = (1:3:0), \quad P_3 = (0:1:1), \quad P_4 = (1:-1:7), \quad P_5 = (1:1:1). \tag{5.4}$$

We can evaluate its determinant and rank as follows.

```
sage: P=[[0,0,1],[1,3,0],[0,1,1],[1,-1,7],[1,1,1]]
sage: f=M(P).det(); f # gives a unique conic!
-96*x^2 + 62*x*y - 10*y^2 + 34*x*z + 10*y*z
```

One way to visualise a projective curve is to set one of the variables equal to 1 and plot in the plane of the other two variables. Here's how to fix $x = 1$ and plot in the (y, z) plane.

```
sage: pf(a,b)=f(1,a,b) # to plot in the (y,z)-plane with x=1
sage: range=(-50,50)
sage: ip=implicit_plot(pf==0,range,range); ip  # displays plot
```



Exercise 5.2. *Figure 5.2 shows $f(1, y, z) = 0$ as in the example above, with the points $(1 : 3 : 0)$, $(1 : -1 : 7)$ and $(1 : 1 : 1)$ superimposed. The points $(0 : 0 : 1)$ and $(0 : 1 : 1)$ are not shown on the plot. Explain how these points are related to the asymptotes of the plot.*

Exercise 5.3. *Find a conic $f(x, y, z) = 0$ passing through the points*

$$P_1 = (0:0:1), \quad P_2 = (0:1:0), \quad P_3 = (0:1:1), \quad P_4 = (1:-1:7), \quad P_5 = (1:1:1).$$

Exercise 5.4. *Use the `implicit_plot` function to plot the conic you found in Exercise 5.3 in the (x, y) plane (setting $z = 1$). Describe the nature of the conic.*

Exercise 5.5. *Factorize the conic you found in Exercise 5.3 and explain how the result relates to the plot in Exercise 5.4.*

Exercise 5.6. *Evaluate* $\det M(x, y, z)$ *for the points*

$$P_1 = (0:0:1), \quad P_2 = (0:1:0), \quad P_3 = (0:1:1), \quad P_4 = (0:-1:7), \quad P_5 = (1:1:1).$$

Does there exist a unique conic passing through these points?

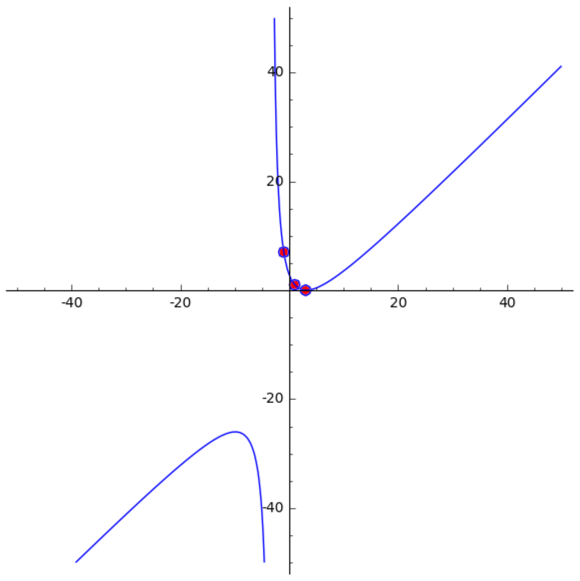


Figure 5.2: The plot of $f = 0$ with 3 visible points.

5.5 Chasles-Cayley-Bacharach theorem and its special cases

The sizes of the matrices in (5.2) and (5.3) are merely the dimensions of the vector spaces $\mathbb{K}_d[x, y, z]$ for $d = 1, 2$. One could (naively) expect this pattern to carry on for $d > 2$, e.g. for $d = 3$ one has $\mathbb{K}_3[x, y, z]$ of dimension 10, and thus one would expect that 9 points P_j determine a projective plane *cubic* (i.e. a curve of degree 3) uniquely, with a 10×10 determinant (5.5) specifying whether a given 10 points lie on such a curve.

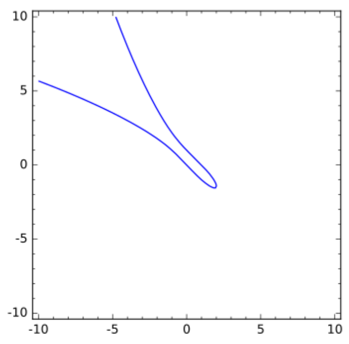
$$M(x, y, z) := \begin{pmatrix}
x_1^3 & y_1^3 & z_1^3 & x_1y_1^2 & x_1z_1^2 & y_1z_1^2 & x_1^2y_1 & x_1^2z_1 & y_1^2z_1 & x_1y_1z_1 \\
x_2^3 & y_2^3 & z_2^3 & x_2y_2^2 & x_2z_2^2 & y_2z_2^2 & x_2^2y_2 & x_2^2z_2 & y_2^2z_2 & x_2y_2z_2 \\
x_3^3 & y_3^3 & z_3^3 & x_3y_3^2 & x_3z_3^2 & y_3z_3^2 & x_3^2y_3 & x_3^2z_3 & y_3^2z_3 & x_3y_3z_3 \\
x_4^3 & y_4^3 & z_4^3 & x_4y_4^2 & x_4z_4^2 & y_4z_4^2 & x_4^2y_4 & x_4^2z_4 & y_4^2z_4 & x_4y_4z_4 \\
x_5^3 & y_5^3 & z_5^3 & x_5y_5^2 & x_5z_5^2 & y_5z_5^2 & x_5^2y_5 & x_5^2z_5 & y_5^2z_5 & x_5y_5z_5 \\
x_6^3 & y_6^3 & z_6^3 & x_6y_6^2 & x_6z_6^2 & y_6z_6^2 & x_6^2y_6 & x_6^2z_6 & y_6^2z_6 & x_6y_6z_6 \\
x_7^3 & y_7^3 & z_7^3 & x_7y_7^2 & x_7z_7^2 & y_7z_7^2 & x_7^2y_7 & x_7^2z_7 & y_7^2z_7 & x_7y_7z_7 \\
x_8^3 & y_8^3 & z_8^3 & x_8y_8^2 & x_8z_8^2 & y_8z_8^2 & x_8^2y_8 & x_8^2z_8 & y_8^2z_8 & x_8y_8z_8 \\
x_9^3 & y_9^3 & z_9^3 & x_9y_9^2 & x_9z_9^2 & y_9z_9^2 & x_9^2y_9 & x_9^2z_9 & y_9^2z_9 & x_9y_9z_9 \\
x^3 & y^3 & z^3 & xy^2 & xz^2 & yz^2 & x^2y & x^2z & y^2z & xyz
\end{pmatrix}, \tag{5.5}$$

$$f(x, y, z) := \det M(x, y, z)$$

The Sage code to create and draw cubic curves using (5.5) is pretty similar to the case for the conic.

```
sage: R.<x,y,z>=QQ[]
sage: v2=vector([x^3,y^3,z^3,x*y^2,x*z^2,y*z^2,x^2*y,x^2*z,y^2*z,x*y*z])
sage: def M(P): return matrix(map(lambda p: v2(p[0],p[1],p[2]), P)+[v2])
sage: P
   =[[0,0,1],[0,1,0],[1,0,0],[1,-4,3],[1,1,-1],[0,1,1],[1,0,1],[1,-1,1],[1,1,0]]

sage: f=M(P).det(); f.factor() # irreducible cubic
(4) * (-36*x^2*y + 36*x*y^2 - 34*x^2*z + 70*x*y*z - y^2*z + 34*x*z^2 + y
   *z^2)
sage: pf(a,b)=f(1,a,b) # to plot in the (y,z)-plane with x=1
sage: range=(-10,10)
sage: ip=implicit_plot(pf==0,range,range); ip  # displays plot
```



Almost surely (i.e. if chosen randomly) 9 distinct points $P = \{P_1, \ldots, P_9\}$ will define a unique cubic curve. On the other hand, if 4 of the P_j are collinear, it turns out that the cubic must be the union of this line and a conic[9]. The question of existence of a unique C on P is thus reduced to the question of existence of a conic on the remaining 5 points of P, which we already considered. In a similar vein, if 7 of the points lie on a conic, it turns out that the cubic must be the union of this conic with a line, again reducing to the previous case. Therefore the interesting remaining case is where no 4 points of P are collinear and where no 7 points of P lie on a conic.

Under these two assumptions, we might expect that P will give us a unique cubic $f = 0$ with f given by (5.5). However, this turns out to fail in a spectacular fashion when there exist two triples of collinear points. Let's consider the construction of 9 points illustrated in Figure 5.3.

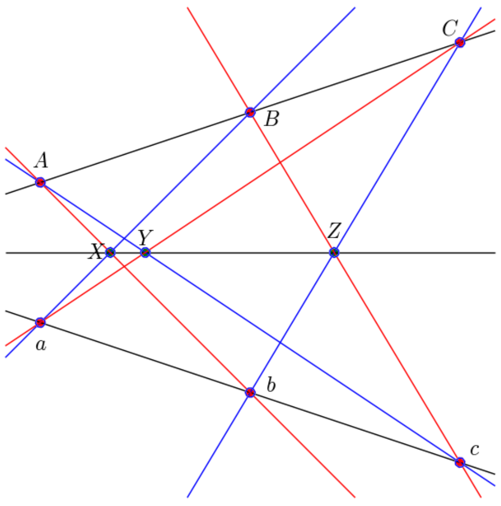


Figure 5.3: A configuration of 9 points.

Let $\{A, B, C\}$ and $\{a, b, c\}$ be two triples of collinear points in the projective plane, and define the points X, Y and Z to be

$$X := Ab \cap aB, \quad Y := Ac \cap aC, \quad Z := bC \cap Bc,$$

where $\cap$ denotes the intersection of two line segments.

More specifically, let us take

$$A = (-2 : 0 : 1), \quad B = (0 : 0 : 1), \quad C = (1 : 0 : 1)$$

[9] This follows for instance from Bezout's theorem which we will mention later.

and

$$a = (0 : 3 : -1), \quad b = (4 : 3 : 1), \quad c = (2 : 3 : 0).$$

It can be shown using (5.2) that both $\{A, B, C\}$ and $\{a, b, c\}$ are collinear (check it!).

Exercise 5.7. *Find the projective points X, Y and Z. You may use pen-and-paper calculation if you wish.*

Exercise 5.8. *Find an equation for each of the red lines in Figure 5.3. Hence find a cubic passing through all 9 points. Do the same for the blue lines and find a different cubic passing through the 9 points.*

Exercise 5.9. *Use Sage to show that X, Y and Z are also collinear, and hence determine a third cubic passing through the 9 points.*

The observation that X, Y and Z are also collinear is true in general for such a setup, and is the celebrated *Pappus Theorem*, which dates back to the 4th century AD.

Exercise 5.10. *Show that $\det M(x, y, z) = 0$ for these 9 points by explicit computation according to (5.5).*

So here is an example in which $\det M(x, y, z) = 0$. This happens because the cubic is not uniquely determined. Or to put it another way, in this case 8 points impose the same conditions as 9! There is a remarkable result which generalises this observation, which is due to Chasles (1837)[10].

Theorem 1. *Let two plane projective cubic curves intersect in exactly 9 distinct points $P_1, \ldots,$ P_9. Then any cubic curve containing $P_1, \ldots, P_8$ will also contain P_9.*

Figure 5.4, which is produced by Sage running the code below, illustrates the result. It shows four cubic curves, drawn in blue, green, orange and brown, which all share eight points of intersection, namely

$$P_1 = (1 : -1 : 7), \quad P_2 = (1 : 5 : -1), \quad P_3 = (1 : -1 : 0), \quad P_4 = (1 : 1 : 0),$$
$$P_5 = (1 : 0 : 2), \quad P_6 = (1 : 0 : 5), \quad P_7 = (1 : -1 : 3), \quad P_8 = (1 : 1 : 3),$$

and then we provide different ninth points to determine the cubic in each case. Some of the points of intersection are shown in Figure 5.4. All four cubic curves intersect in a 9th (circled) point P_9 (as they should by Theorem 1).

```
R.<x,y,z>=QQ[]
v2=vector([x^3,y^3,z^3,x*y^2,x*z^2,y*z^2,x^2*y,x^2*z,y^2*z,x*y*z])
def M(P):
    return matrix(map(lambda p: v2(p[0],p[1],p[2]), P)+[v2])

# the following line contains the 8 points P_1,...,P_8.
P=[[1,-7,-1],[1,5,-1],[1,-1,0],[1,1,0],[1,0,2],[1,0,5],[1,-1,3],[1,1,3]]

range2=(-1,4)
range1=(-2,2)
G=Graphics()
var('a b');
for p,col in [([1,0,1],'blue'),([1,1,-7],'green'),([0,1,0],'orange')
    ,([0,0,1],'brown')]:
    P1 = P+[p]
```

[10]For historical and other reasons, Theorem 1 is often called the Cayley-Bacharach Theorem. However, Cayley published, 6 years afterwards, an erroneous generalisation of Chasles' result to curves of higher degree, which was fixed by Bacharach in 1873.

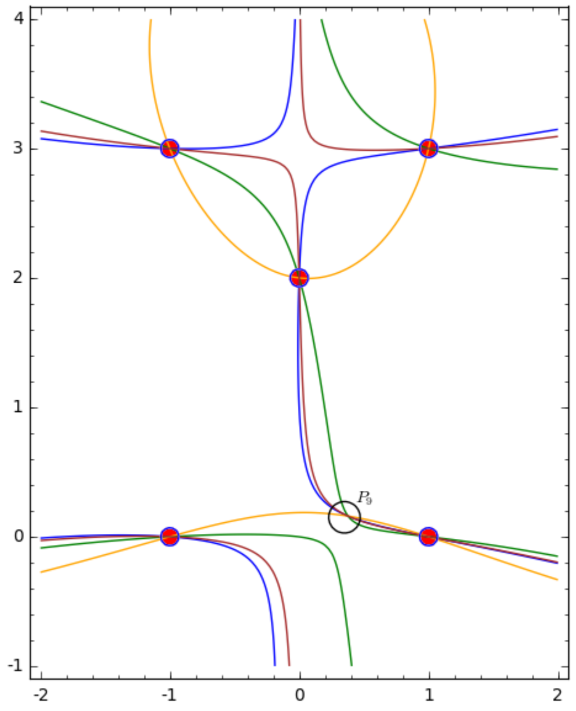


Figure 5.4: The mysterious P_9. The red points are 5 of the 8 prescribed intersection points from P (the remaining 3 are outside the plotted area).

```
    f1=M(P1).det()
    pf1(a,b)=f1(1,a,b) # to plot in the (y,z)-plane with x=1
    G+=implicit_plot(pf1==0,range1,range2,color=col)
for p in P:
  if p[0]!=0:                    # skip points at infinity
     pp(a,b)=(a-p[1])^2+(b-p[2])^2-0.005
     G+=implicit_plot(pp,range1,range2,fill=True,fillcolor='red')
pp(a,b)=(a-0.35)^2+(b-0.15)^2-0.015
G+=implicit_plot(pp,range1,range2,color='black')
G+=text("$P_9$", (0.5,0.3), color='black'); G
```

Exercise 5.11. *Find the exact coordinates of the 9th point of intersection on Figure 5.4. Hint: use Sages's* `resultant` *function, which allows to eliminate variables from a system of polynomial equations, and then factor the obtained univariate polynomial.*

5.6 Sage submission notes

To submit this project, save (download) your work from cloud.sagemath.org as a file

`projectC_12345678.sagews`

Replace 12345678 with your candidate number.

Also print your worksheet to a PDF file: preferably by using the print icon next to the green "Save" button.

Place both files inside a .zip file as described in the MATLAB -specific instructions and submit normally.

NOTE: simply sharing your cloud.sagemath.org project with your instructor won't work (Oxford grading must be anonymous).